VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra Informatiky a výpočetní techniky

# Komponenta pro vizualizaci grafových struktur
# Graph Visualization Component

2012 Ondřej Svačina

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Ondřej Svačina**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Komponenta pro vizualizaci grafových struktur
Graph Visualization Component

Zásady pro vypracování:

Cílem práce je implementace vizualizačního nástroje, který bude sloužit k zobrazení grafové struktury nad vybranou kolekcí. Vybranou technologií bude Microsoft .NET, Silverlight.

1. Přehled existujících nástrojů a technik pro vizualizaci informace s bližším zaměřením na vizualizaci grafů.
2. Návrh aplikace ve vybrané technologii.
3. Implementace a vyhodnocení výkonostích testů.
4. Zhodnocení dosažených cílů a provedených experimentů.

Seznam doporučené odborné literatury:

Nick Randolph,Christopher Fairbairn, Professional Windows Phone 7 Application Development: Building Windows Phone, ISBN-13: 978-0470891667, Wrox; 1 edition (November 9, 2010)

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Gajdoš, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

# Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Ve Studénce Dne: 26.4.2012 Podpis: 

# Poděkování

Rád bych poděkoval panu Ing. Petrovi Gajdošovi, Ph.D. za trpělivost, odbornou pomoc a konzultaci při vytváření této práce.

# Abstrakt

Cílem této práce je seznámit čtenáře se základními způsoby zobrazení dat a poté navrhnout příklad programu pro zobrazení dat. V první části práce se čtenář dozví základní informace o zobrazení dat. Tato část se zabývá základními způsoby vizualizace grafů. Mezi popisované způsoby patří dendrogramy a algoritmy pro rozmístění uzlů grafu. V druhé části práce se bude věnovat přehledu technologií. Tato část bude obsahovat informace o novém programovacím modelu Windows Runtime, který v podstatě nahrazuje současný Win32. V této sekci také najdeme informace o technologii Silverlight, ve které byl vytvořen ukázkový program, a o technologii Windows Forms. Třetí část práce se zabývá popisem samotné implementace ukázkového programu pro vizualizaci dat. Zde bude probrána základní architektura programu a datová vrstva. Dále zde bude popsán návrh funkcionality a grafické rozhraní programu. Další bod bude popsání samotné funkcionality. Nakonec budou uvedeny výkonnostní testy pro klíčové vlastnosti programu a jejich stručné vyhodnocení.



# Klíčová slova

Graf, grafická struktura, zobrazení dat, zobrazení, uzel, hrana.



# Abstract

The aim of this work is to introduce reader with ways to display data and then design an example program to display data. In the first part the reader will finds basic information about how to display data. This section covers the basic ways to visualize graphs. Described methods include the dendrograms and algorithms for graph nodes deployment. In the second part will be devoted to an overview of technologies. This section will contain information about the new programming model, Windows Runtime which basically replaces the current Win32. In this section you also find information about Silverlight, which was used to create an example program, and Windows Forms technology. The third part deals with the description of the implementation of an example program for data visualization. Here will be discussed basic architecture and the data layer. There will also be described functionality and program interface. Another section will describe the functionality itself. At the end of the work will be set of performance tests for the key features and brief assessment.



# Key words

Graph, graph structure, data display, display, node, edge.

# Seznam tabulek

# Seznam obrázků

# Obsah

# 1. Úvod

V dnešní době jsou data naprosto nezbytná pro téměř jakoukoli činnost. Téměř celý moderní svět funguje na základě dat. I peníze, základní stavební pilíř moderní společnosti, v bankách existují pouze jako data. Data však mohou být zpracovány různými způsoby. Můžou být například zobrazena jako text formou obyčejné tabulky, nebo graficky formou různých typů grafů. Právě pro druhý případ existují metody pro pohodlné a přehledné zobrazení těchto grafů. Každá tato metoda má snahu zobrazit graf takovým způsobem, aby vynikla některá z vlastností grafu. Mezi tyto metody patří dendrogramy a algoritmy pro rozmístění uzlů grafu (též známe jako layout algoritmy). Teoretická část práce se věnuje právě těmto metodám.

Cíl této práce je tedy seznámit čtenáře se základními metodami pro přehledné zobrazení dat a vytvořit v prostředí Silverlight ukázkový program pro zpracování grafové kolekce dat a následné zobrazení této kolekce formou nesměrového váženého grafu. Program by měl zvládat s tímto grafem manipulovat a provádět základní operace. Více na toto téma v kapitole implementace.

Prostředí Silverlight bylo zvoleno vedoucím práce z důvodů snadné dostupnosti výsledného programu na webu, bohaté podpoře grafického rozhraní, podpoře různých platforem a prohlížečů a Silverlight obsahuje vhodné třídy a komponenty pro vykreslování obrazců, v našem případě grafů. Další důvody a více informací o technologii Silverlight se dočtete v podkapitole Silverlight v kapitole přehled technologií.

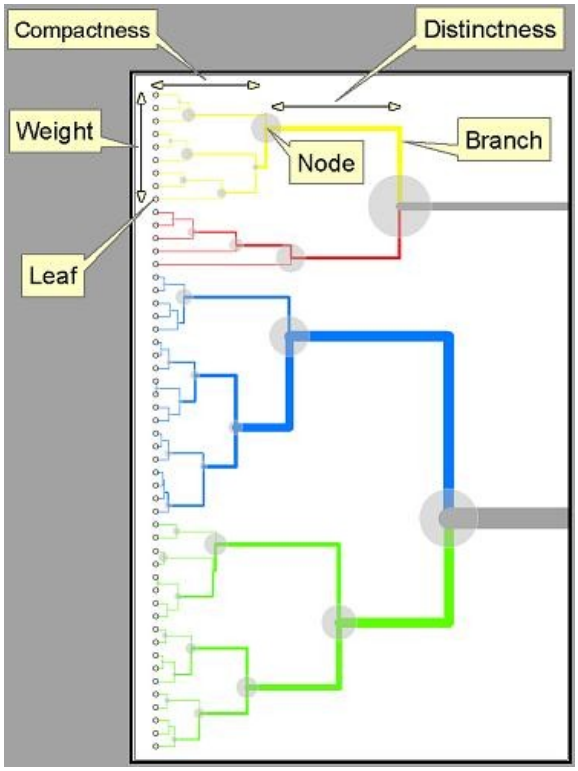
V závěru práce se nachází výkonnostní testy, jež vyhodnotí vytvořený program, zda byl navržen tak, aby mohl v praxi pracovat i velkými datovými kolekcemi.

# 2. Teorie

Tato část se zabývá základními způsoby zobrazování grafů. První části se zabývá Dendrogramy. Dendrogramy jsou vzhledem podobné binárním stromům, které mají za úkol přehledně zobrazit shluky uzlů. Druhá část se zabývá metodami plošného rozmístění uzlů grafu. Tyto metody se snaží zobrazit data co nejpřehledněji podle daných kritérií a zároveň co nejužitečněji vyplnit prostor. V této části se také nachází popis 5 nejčastěji používaných metod. Na konec první část také obsahuje teorii k metodám hledání podobností. Tato část se věnuje hlavně podobnosti mezi texty. Tyto metody často vytvářejí kolekce dat pro vážené grafy.

## 1. Dendrogramy

Dendrogram je stromový diagram, jenž se používá pro grafické znázornění výsledku shlukové analýzy. Větve diagramu reprezentují shluky získané během jednotlivých kroků shlukové analýzy. Každý tento krok je reprezentován sloučením dvou větví v jednu. Jakmile je algoritmus shlukové analýzy spuštěn, uživatel může rozhodnout, kolik shluků chce pozorovat. Tomuto se také říká prořezávání dendrogramu.




**1 Dendrogram Zdroj: Http://edndoc.esri.com/arcobjects/8.3/Samples/Analysis%20and%20Visualization/Cluster%20Analysis/CLUSTERANALYSIS.htm.**

Na obrázku byly zvoleny 4 skupiny, reprezentovány různými barvami. Rozhodnout o počtu zobrazených shluků nám může pomoct pohled na dendrogram. Z dendrogramu vyčteme 3 klíčové informace, a to váha – hrubé procento prvků, jež spadají do daného seskupení, kompaktnost – jak moc

jsou si jednotlivé prvky shluku podobné a odlišnost – jak moc se liší jeden shluk od nejbližšího souseda.

Váha jednotlivých shluků je dána počtem koncových uzlů (leaf), které daná větev (branch) obsahuje. Protože všechny koncové uzly jsou pravidelně rozmístěné s totožnou výškou, váha shluku je také procento výšky diagramu, jež zabraly koncové uzly daného shluku.

Kompaktnost shluku představuje minimální vzdálenost, během které shluk vznikne. Tato hodnota se odečítá pomocí horizontální osy. Pokud shluk obsahuje pouze jeden prvek, kompaktnost je rovno 0, proto jsou všechny koncové uzly seřazeny úplně vlevo. Relativní kompaktnost u žlutého shluku na obrázku může být určena pohledem na bod, ve kterém se všechny větve daného shluku spojují dohromady a relativní vzdáleností z tohoto bodu na levou stranu diagramu.

Odlišnost shluku je vzdálenost na vodorovné ose z bodu, kdy vzniká shluk do bodu, kdy se tento shluk spojuje s dalším shlukem.

# 2. Plošné rozmístění uzlů grafu

Algoritmy pro plošné rozmístění uzlů rozhodují o umístění jednotlivých uzlů a hran grafu tak, aby výsledný graf esteticky a přehledně reprezentoval data. Existuje spousta různých algoritmů pro rozmístění uzlů, podle typu aplikace a zobrazovaných dat. Například požadavky na rozmístění uzlů stromového grafu můžou být naprosto rozdílné od grafu směrového. I když je výsledné rozmístění uzlů vnímáno subjektivně a mělo by být doladěno tak, aby souhlasilo s požadavky jedince, některé kritéria jako je snaha minimalizovat počet křížení a ohýbání hran, zobrazování podobností v grafech a minimalizování celkové velikosti grafu jsou běžné pro všechny způsoby rozmístění uzlů.

Stejně jako v jiných oblastech, i zde je velká mezera mezi teorií a skutečnou praxí. Většina teoretických poznatků se ve výsledku týkají pouze silně omezených grafů, takže se tyto poznatky v praxi nedají využít u většiny grafů. V současnosti je snaha o vývoj algoritmů, jež se zaměřují na tyto cíle: řešení by mělo být platné pro jakékoli grafy, být nenáročné na prostor, časovou nenáročnost a co nejvyšší možnosti úprav. Více informací o algoritmech pro rozmístění uzlů v knize [2].

V mé práci používám jeden ze silově závislých rozmístění – Fruchterman Reingold Layout algoritmus. Tento algoritmus jsem si zvolil, protože výsledek je velmi přehledný, algoritmus není složitý a pro mé potřeby je časová náročnost dostatečná. Více v silově závislých rozmístění.

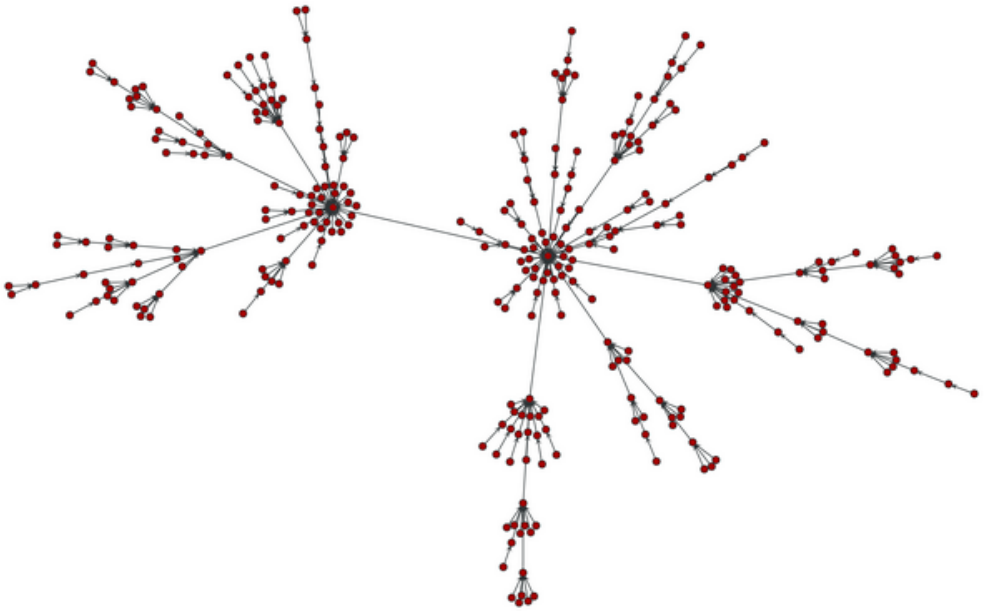
Dále následuje výčet nejznámějších algoritmů a krátký popis s příkladem obrázku výsledného rozmístění uzlů.

## Silově závislé rozmístění

Cílem tohoto algoritmu je vykreslit graf co nejvíc esteticky. Uzly jsou umístěny do prostoru tak, že hrany jsou zhruba stejně velké a zároveň se hrany kříží co nejméně. Algoritmus tohoto dosahuje přidělením hodnoty síly uzlům a hranám. Poté se budou jednotlivé uzly od sebe navzájem přitahovat či odpuzovat danou silou, v závislosti na nejkratší cestě mezi těmito uzly. Tento krok se opakuje, dokud nedojde ke stavu rovnováhy. To je když se pozice uzlů při další iteraci již nemění. V tento moment je graf vykreslen.

Výhody tohoto algoritmu jsou velmi kvalitní výsledné rozmístění uzlů pro středně velké grafy, algoritmus lze snadno upravit dle svých potřeb a jednoduchost na naprogramování.

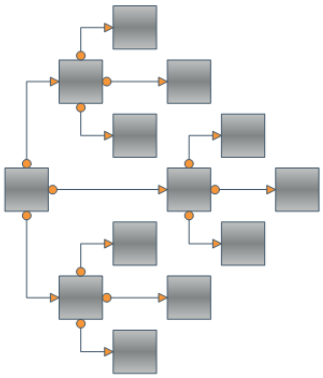
Nevýhoda je vysoká časová náročnost. Sice existují způsoby, kterými lze tuto náročnost mírně snížit, ale stále se bude jednat o časově velmi náročný algoritmus. Jedná se zhruba o $O(n^2)$, kde n je počet uzlů v grafu. Více informací o tomto rozmístění v knize [10].



**2 Graf rozmístěný pomocí silového rozmístění.**
**Zdroj: http://projects.skewed.de/graph-tool/doc/draw.html.**

## Kolmé rozmístění

Toto rozmístění umožňuje hranám grafu být umístěný vodorovně či svisle podél os rozložení grafu. Tento algoritmus se provádí v několika fázích, kde se v první fázi pro vytvoření návrhu rozmístění uzlů nahrazuje křížení hran uzlem. Jakmile je rozhodnuto o umístění uzlů, v druhé fázi se uzly natočí tak aby došlo k minimálnímu počtu křížení. Ve třetí fázi se snažíme minimalizovat celkovou velikost rozložení uzlů. Tento algoritmus se v praxi moc nepoužívá. Více informací v knize [6].



**3 Graf rozmístěný pomocí kolmého rozmístění.**
**Zdroj: http://www.nevron.com/Products.DiagramFor.NET.WhitePaperOrthogonalGraphLayout.aspx.**

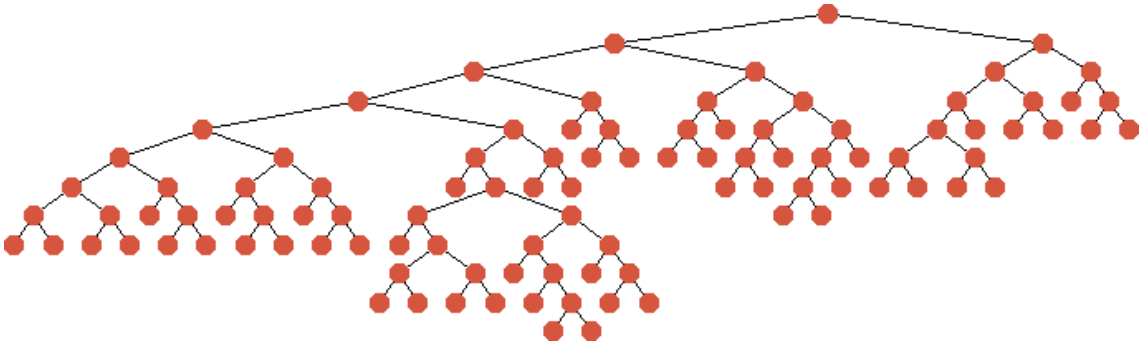
## Stromové rozmístění

Pomocí tohoto algoritmu rozložíme graf do stromu podobného uskupení. Tento algoritmus se používá pro zobrazení stromových datových struktur, jako jsou například soubory a složky v operačním systému Windows. Tyto datové struktury se dělí na kořenové stromy a volné stromy.

Kořenové stromy mají jasně danou hierarchii a obsahují kořenový uzel. Pokud zobrazujeme daný strom, musíme zachovat a zvýraznit hierarchii. Proto se kořen vykresluje obvykle nahoře a každý stupeň potomků se vykresluje o úroveň níže. Další metoda je vykreslit kořen doprostřed a potomky vykreslovat ve zvětšujících se soustředných kružnicích se středem v kořenu.

Volné stromy naopak žádnou strukturu mimo samotné propojení uzlů neobsahují. Tudíž se graf nemůže vykreslit hierarchicky a musí se použít jiné metody.

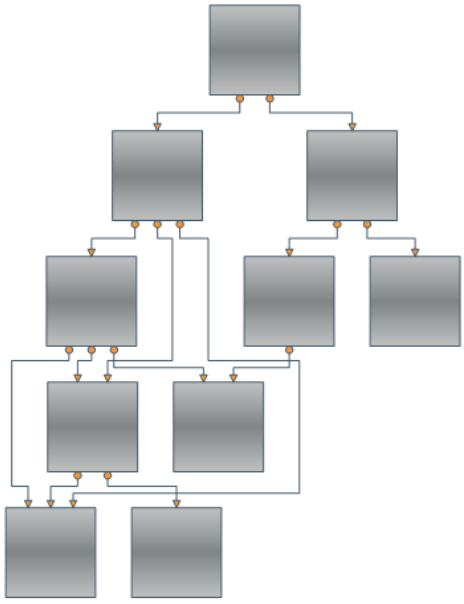
Více informací o stromovém rozmístění lze nalézt zde [7].



**4 Strukturovaný binární graf rozmístěný pomocí stromového rozmístění.**
**Zdroj: http://www.informatik.uni-koeln.de/ls_juenger/research/vbctool.**

## Vrstvené rozmístění

Tato metoda se také nazývá po svém autorovi Kozo Sugiyama Sugiyama-style graph drawing a používá se u směrových, necyklických grafů jako jsou grafy závislostí. V této metodě se první uzly rozdělí do horizontálních vrstev tím způsobem, aby většina hran šla z vrchní vrstvy do spodní. Poté jsou uzly v každé vrstvě uspořádány tak, ať dojde k co nejméně křížení hran. Časová náročnost této metody je $O(m * n)$, kde m je počet hran a n je počet uzlů. Metoda lze být dále zjednodušena na téměř lineární závislost. Více o tomto rozmístění zde [8].



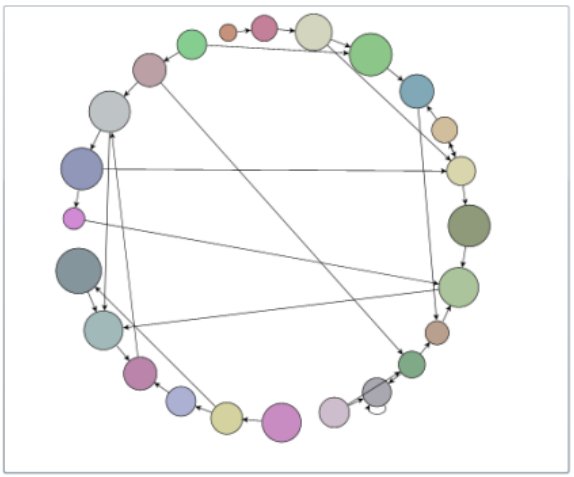
**5 Graf rozmístěný pomocí vrstveného rozmístění.**
**Zdroj: http://helpdotnetvision.nevron.com/UsersGuide_Layouts_Layered_Graph_Layout.html.**

## Kruhové rozmístění grafu

Toto rozmístění se snaží rozmístit uzly do jednoho či více kruhů a není vhodný pro grafy s uzly vysokých stupňů.

Jedná-li se o rozložení do jednoho kruhu, tak jsou uzly rozděleny tak, aby sousední uzly v kružnici byly zároveň uzly spojené hranou. Zároveň, pokud uzel již má určeny oba sousedy, je každý další uzel spojený hranou posunut v kruhu co nejvíce na druhou stranu.

Jedná-li se o rozložení do více kruhů, jsou uzly rozmístěny takovým způsobem, aby byly uzly rozděleny do skupin. Uzly každé této skupiny pak jsou rozmístěny na vlastní kružnici podle způsobu popsaného výše. K tomuto tématu více najdete zde [9]



**6 Graf vytvořený pomocí kruhového rozmístění.**
**Zdroj: http://www.nwoods.com/components/silverlight-wpf/goxam-layout.htm.**

# 3. Metody hledání podobnosti

Metody hledání podobnosti textů vracejí hodnotu v intervalu <0,1>, kde 0 znamená absolutní neshodu, zatímco 1 znamená absolutní shodu. Metody hledání podobností se dají obecně rozdělit na dva typy.

První typ nejdříve spočítá jistou hodnotu pro oba texty a následně tyto hodnoty porovná dle vzorce $x = \frac{Min\ a,b}{Max\ a,x}$. Do prvního typu patří metody porovnávání počtu výskytu tokenů, porovnávání směrodatných odchylek výskytu tokenů a porovnávání počtů různých identifikátorů.

Druhý typ tvoří řetězcové metriky. Jedná se o metody, které počítají různými způsoby vzdálenosti dvou řetězců, a poté tuto hodnotu upraví vzhledem k délce řetězců. Do druhého typu patří metody Levenshteinova vzdálenost, Hammingova vzdálenost a nejmenší společný podřetězec.

Metody hledání podobnosti mezi obrázky se řeší převedením obrázků na text či kód vyjadřující obrázek, a poté se tyto texty porovnají pomocí speciálních metod. Více v knize [5].

Dále jsou uvedeny některé metody hledání podobnosti textů.

## Porovnání počtu výskytu tokenů

Pro použití této metody je třeba jako parametr dodat, o výskyt kterého tokenu se zajímáme. Tato metoda pro oba texty spočítá výskyt daného tokenu a podle výše uvedeného vzorce tuto hodnotu

porovná. Tato metoda má spolehlivé výsledky a časová složitost metody se počítá podle vzorce $O(Max\ m,n)$ , kde m, n jsou velikosti porovnávaných textů. Více informací v knize[1].

## Levenshteinova vzdálenost

Tato metoda počítá nejmenší počet operací, aby se z řetězce a stal řetězec b. Mezi tyto operace se počítá smazání, přidání nebo nahrazení znaku. Tato metoda je velice spolehlivá avšak náročná na výpočet. Časová složitost metody se počítá podle vzorce $O(m * n)$. Tato metoda se často používá pro kontrolu překlepů. K této metodě najdete více v knize [13].

## Nejdelší společný podřetězec

Tato metoda počítá nejdelší společný řetězec zadaných vstupů. Čím je tento řetězec delší, tím je větší podobnost mezi soubory. Tato metoda je opět velice spolehlivá, ale časová složitost je vysoká dle vzorce $O(m * n)$. Více o této metodě v knize [11].

## Hammingova vzdálenost

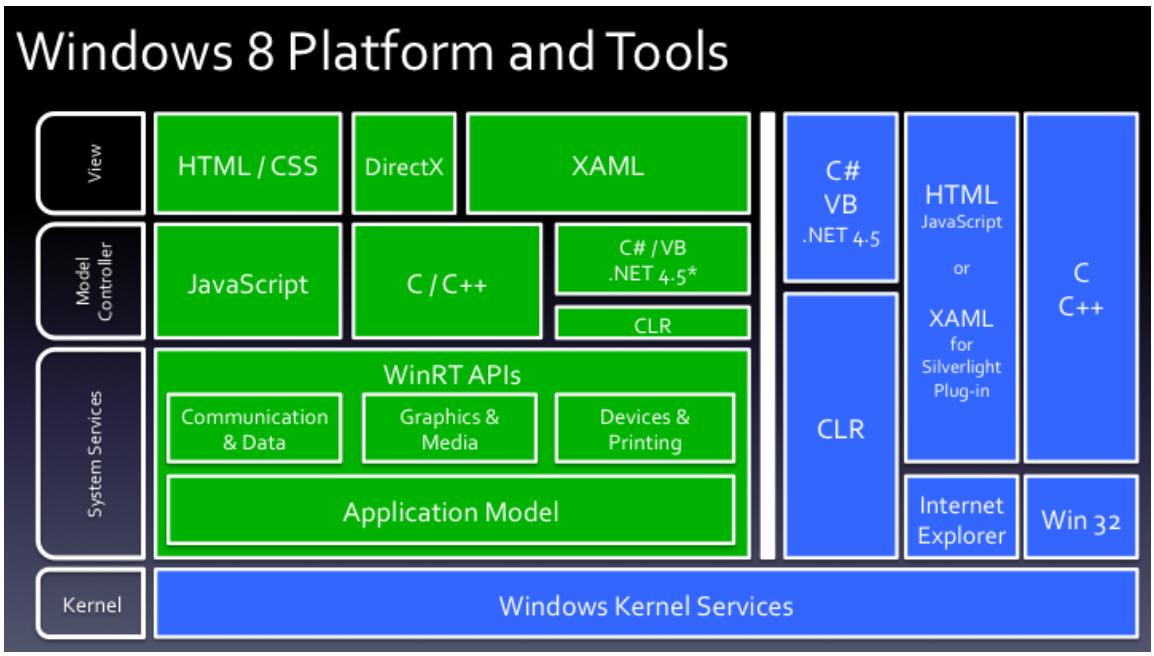
Tato metoda počítá počet shodných znaků na totožných místech v řetězci. Časová složitost metody je lineární, avšak spolehlivost metody je nízká. Pokud na začátek řetězce přidáme libovolný znak, pozice znaků v řetězci se posunou a výsledná podobnost vyjde mizivá. Více v knize [12].

# 3. Přehled technologií

Microsoft přichází v nových Windows 8 s novým programovacím modelem, Windows runtime zkráceně WinRT. WinRT dovoluje používat HTML ke stejnému účelu jako jazyk XAML a Javascript k ke stejnému účelu jako C#. Pro Aero se stále používá Win32, ale pro Metro je nutné použít WinRT.

WinRT oproti předchůdcům není založeno na .NET, ale na upravené verzi COM. Avšak vývojáři nemusí znát složitosti COM, aby mohli psát kód pro WinRT. Místo toho si Microsoft vypůjčil formát metadat z .NET. Ve výsledku je tedy možné používat C/C++ i .NET. Microsoft také použil jazyk XAML, jazyk používaný hlavně v technologii Silverlight, pro tvorbu uživatelského rozhraní ve WinRT. Aby tohoto docílil, byl XAML přepsán na přirozený kód, aby nevznikly žádné závislosti na .NET.

WinRT je implementované v C++, tudíž aplikace tvořené v C++ budou pro WinRT nejefektivnější. Kód pro WinRT se píše v jazyce C++/CX (C++ component extension), což je C++ s rozšířenou syntaxí. Třídy ve WinRT jsou v podstatě upravené COM objekty.




**7 Architektura Windows 8.**
**Zdroj: http://kodierer.blogspot.com/2011/09/welcome-to-zombieland-metro-style-land.html.**

Dále se tato část práce věnuje již zmíněné technologii Silverlight, jejíž součástí je také jazyk XAML Technologie Silverlight byla vybrána pro tvorbu ukázkového programu, proto je tomuto tématu věnováno více prostoru. Nakonec se v této části nachází také podkapitola Windows Forms. Tato technologie je dnes již zastaralá, ale většina moderní technologie z Windows Forms vychází.

# 1. Silverlight

Microsoft Silverlight je zásuvný modul určený pro různé prohlížeče, platformy a dokonce zařízení. Tento zásuvný modul slouží k poskytování nové generace interaktivních webových aplikací postavených na zkušenostech z .net aplikací. Silverlight je viděn jako alternativa k Adobe's Flash, široce rozšířenému zásuvnému modulu, jenž umožňuje podobnou funkcionalitu jako Silverlight.

Silverlight aplikace se integrují do již existujících webových aplikací, včetně ASP .NET AJAX. Silverlight může komunikovat s jakoukoli AJAX aplikací jak na straně klienta, tak na straně serveru.

Silverlight je používán k přidávání multimediálních prvků na webové stránky. Mezi tyto prvky například patří fotky a vestavěné animace. Dále lze používat a vytvářet interaktivní prvky jako je chat, online hlasování a vyskakovací okna. Silverlight dále podporuje velké množství vysoce kvalitních video a audio formátů. Mezi nativně podporované formáty patří VC-1/WMA, MPEG-4 a audio formát AAC.

Aplikace tvořená v Silverlight je celá uložena do XAP souboru a vložena na web. V tomto souboru jsou pouze výsledné dll soubory, jež obsahují samotný kód a všechny ostatní nutné prvky jako jsou obrázky či zvuky. Pokud uživatel vstoupí na stránku s touto aplikací a má nainstalovaný Silverlight, tento soubor se nahraje pomocí klasického požadavku get a program se spustí.

Silverlight je navržený tak, aby překonal limity HTML a umožnil vytvářet graficky lepší a více interaktivní aplikace. Avšak pro Silverlight aplikace, stejně jako pro obyčejné webové stránky, platí jisté restrikce. Například ukládat nebo načítat data se dají jen ze speciálně vyhrazené oblasti tzv. „isolated storage". Existují však způsoby jak toto omezení obejít.

Silverlight může být používán několika nejznámějšími operačními systémy a prohlížeči, ale ne všemi. Mezi operační systémy podporující Silverlight především patří Microsoft Windows, Linux a Macintosh. Mezi prohlížeče podporující Silverlight patří Microsoft Internet Explorer, Safari, Opera, Firefox a Chrome.

Silverlight se také používá jako vývojový nástroj pro Windows Phone 7, což je operační systém pro mobilní zařízení od firmy Microsoft. Programy vytvořené v Silverlight pro Windows Phone mohou být poté umístěny na Windows Phone Marketplace, kde uživatelé mohou tyto aplikace kupovat pro své telefony.

Od vydání Silverlight 1.0 v roce 2007 bylo vydáno již 5 verzí. Momentálně nejaktuálnější verzí je Silverlight 5, jenž oproti předchozí verzi obsahuje spoustu novinek, například zlepšení srozumitelnosti textu, zlepšená kontrola videí a vylepšená grafika.

Více informací k nalezení v knize [3].


**8 Ukázková aplikace vytvořená pomocí Silverlight.**
**Zdroj: http://www.arcdata.cz/aktuality/aktuality-detail/?contentId=120830.**

## XAML

Uživatelské rozhraní Silverlight aplikací se obvykle tvoří pomocí jazyku XAML. XAML je zkratka pro extensible application markup language. Jedná se o deklarativní jazyk založený na XML. XAML umožňuje deklarativním přístupem vytvářet bohaté uživatelské rozhraní. Každý element v XAML je reprezentován stejnojmennou třídou, jenž se nachází ve jmenném prostoru System.Windows.Controls. Jazyk XAML slouží k usnadnění zápisu. Vše, co lze udělat pomocí něj, tak můžeme zapsat i v jazycích C# nebo Visual Basic .NET.

Další klíčovou myšlenkou jazyka XAML je oddělení tvorby uživatelského rozhraní od tvorby programové části. Tímto oddělením je umožněna paralelní práce vývojářů a návrhářů uživatelského rozhraní. Uživatelské rozhraní se často tvoří pomocí nástroje Microsoft Expression Blend, jenž umožňuje tvořit uživatelské rozhraní i lidem, jenž neumí dobře programovat.

### Výhody

- Díky Silverlight se můžou vývojáři při vývoji aplikací pro prohlížeče soustředit na jedno konkrétní prostředí, místo vytváření různých verzí pro jednotlivé prohlížeče.
- Snadná instalace Silverlight.
- Silverlight interpretuje XAML přímo. XAML je přiložen do XAP souboru. XAP soubor je v podstatě ZIP, tudíž vyhledavače mají možnost indexovat vnitřní text aplikace.
- Isolované uložiště umožňuje Silverlight aplikacím přístup do lokálních souborů.
- Silverlight podporuje několik druhů programovacích jazyků. Mezi nimi jsou například C# a Visual Basic.
- Velké množství ovládacích prvků.
- Snadná tvorba snad jakékoli animace.

### Nevýhody

- Největší nevýhodou Silverlight je rozšířenost. Zatímco největší konkurent Flash je podle nejnovějších průzkumů nainstalován v 95% počítačů, Silverlight je nainstalován pouze v 65%.
- Silverlight není na trhu tak dlouho jako konkurence, tudíž nemá tak rozšířenou komunitu. Díky tomu je na internetu daleko více návodů a tipů pro konkurenční platformy než pro Silverlight. Také to znamená, že Silverlight oproti konkurenci nebyl tak dobře otestován.
- Flash je lépe vybaven pro umístění videí na internetové stránky.
- Ačkoli Expression Blend a Visual Studio jsou dobře propojené, je poněkud nepříjemné používat 2 různé, oddělené nástroje.
- Na všech uživatelských počítačích musí být nainstalovaný zásuvný modul Silverlight, aby se aplikace mohla zobrazit.
- Říka se, že Microsoft uvažuje nad ukončením podpory Silverlight.

## 2. Windows Forms

Jedná se o grafické programovací rozhraní pro aplikace (API), které je součástí Microsoft .net Framework. Umožňuje přístup ke klasickým prvkům rozhraní Windows tím, že zabaluje již existující API do tzv. „managed“ kódu.

Windows Forms aplikace jsou událostmi řízené aplikace podporované .net frameworkem. Tyto aplikace stráví většinu času čekáním na činnost uživatele, jako například zmáčknutí tlačítka.

Pomocí Windows Forms se aplikace vytváří podobně jako pomocí Microsoft Foundation Class, avšak Windows Forms neposkytují výchozí aplikační Framework. Windows Forms oproti klasickým

formulářovým aplikacím není závislý na platformě, jedná se o skutečné objektové paradigma pro vývoj, je jednodušší, pohodlnější a můžeme používat různé programovací jazyky platformy .NET.

Každý ovládací prvek ve Windows Forms aplikacích je instancí třídy. Rozložení ovládacích prvků na uživatelském rozhraní a chování těchto prvků jsou řízeny pomocí metod. Windows Forms poskytují širokou škálu ovládacích prvků, jako jsou textová pole, tlačítka, nápisy, tabulky a jiné. Dále je také umožněno vytvářet vlastní ovládací prvky. Windows Forms také umožňují vytvářet a používat třídy pro práci s štětci, barvami, fonty, ikonami a jinými grafickými objekty.

Základem Windows Forms aplikace je formulář, na který umisťujeme ovládací prvky. Každá Windows Forms aplikace musí obsahovat aspoň jeden formulář.

Další vlastností Windows Forms je nastavení aplikace, jež je ukládáno formou xml souboru a umožňuje vytvářet, ukládat a udržovat stavové informace. Tyto informace lze použít k načtení uživatelem preferovaného nastavení, jako jsou pozice nástrojů. Tato nastavení mohou být použita i v budoucích aplikacích.

Více informaci v knize [4].

## Výhody

- Díky svému stáří se dá na internetu najít spoustu návodů, příkladů, dokumentace, vyřešených problémů, knihoven, doplňků a komponent.
- Jednoduchý program se dá vytvořit velmi snadně a rychle.
- Podporuje novější technologii WPF.
- Windows Forms jsou snadné na naučení.

## Nevýhody

- Zastaralost – WPF je v podstatě novější verze Windows Forms.
- Windows Forms již nejsou podporované, provádí se již jen údržba.
- Tvorba aplikací, které budou vypadat skutečně dobře a podle představ vývojáře je velmi časově náročné.
- Neobsahuje pokročilé grafické prvky.

# 4. Implementace

Tato kapitola se věnuje samotnému ukázkovému programu. V této části se nachází informace o architektuře programu, obsahující informace jak byl program navržen. Dále se zde nachází návrh datové vrstvy, ve které je popsáno jakým způsobem program zpracovává a ukládá data. V další části se dočteme o základní funkcionalitě programu, kde je popsáno, co vše program vlastně zvládá. Další část se věnuje popsání GUI, jak uživatel ovládá program a kde se v uživatelském rozhraní co nachází. Předposlední podkapitola se věnuje samotné implementaci. Je zde popsáno, jakým stylem byly naprogramovány nejdůležitější části programu. Nakonec jsou v této kapitole uvedeny výkonnostní testy, obsahující naměřené hodnoty zobrazené formou tabulek a grafu pro základní operace s programem a jejich textové vyhodnocení.

## 1. Architektura programu

Program je vytvořen pomocí platformy Silverlight v programovacím jazyku c#. Program je navržen následovně: Po načtení je XML soubor poslán do datové vrstvy (DataModel). Zde je soubor rozložen na jednotlivá data a uložen do dynamických listů. Jakmile je soubor úspěšně zpracován, odešlou se listy do prezenční vrstvy (View), kde se k těmto datům vytvoří uzly. K uzlům se přiřadí hrany, a pak se uzly i hrany uloží do dynamických listů, jež dávají dohromady výsledný graf. Poté je nutné k vytvořeným uzlům a hranám přiřadit souřadnice na plátně. Proto na graf použijeme Fruchterman Reingold Layout algoritmus. Tento algoritmus vygeneruje vhodné souřadnice pro rozmístění uzlů a tyto získané souřadnice uloží. Dále program pomocí metody setSizes nastaví na základě hodnot hran velikost uzlů a hran a tyto hodnoty opět uloží do grafu. Nyní již má program uložená všechna potřebná data v modelu a data potřebná pro vykreslení ve view, takže program může zahájit vykreslování. Jakmile jsou data vykreslena, zbývá pro plnou funkčnost vytvořit instanci třídy DragNodes, která obstarává veškeré činnosti s uzly, jako je označení uzlu, označení výběrovým oknem více uzlů současně, přidávání jednotlivých uzlů mezi vybrané pomocí klávesy shift a posun všech vybraných uzlů.

## 2. Návrh datové vrstvy

### Model

Nejvýše v datové hierarchii stojí abstraktní třída node. Tato třída obsahuje atributy jméno, id a datum přidání do kolekce. Z této třídy dále dědí další třídy, které reprezentují konkrétní objekty a obsahují atributy potřebné pro tyto objekty. V ukázkovém případě se jedná o vývojáře a software. Dále existuje třída Edge, reprezentující hrany mezi objekty. Třída Edge obsahuje id, hodnotu hrany a odkazy na objekty, které jsou hranou spojeny. Objekty vytvořené z těchto tříd jsou uloženy do dynamických listů ve třídě model. Tyto objekty jsou také pomocí metody SortSubgraphs() rozděleny na jednotlivé podgrafy.

### View

Jakmile chce program zobrazit graf, musí nejprve vytvořit strukturu uzlů a hran. Zde přichází na řadu třída Graph ve view. Tato třída obsahuje dynamický list uzlů a hran ve view. Uzly jsou tvořeny třídou Node. Třída Node obsahuje souřadnice uzlu na plátně, elipsu, jenž graficky reprezentuje uzel, id uzlu, šířku uzlu, výšku uzlu, štětce potřebné pro správné vykreslení uzlu a jiné atributy. Hrany jsou tvořeny třídou Edge. Tato třída obsahuje odkazy na uzly, jež propojuje, váhu hrany, id hrany, objekt Line, jenž graficky reprezentuje hranu a barevný štětec s barvou hrany.

Uzly i hrany ve view i modelu mají totožné id, tudíž pracujeme-li ve view s uzlem s ID 1, pracujeme v podstatě s uzlem reprezentující data v modelu s ID 1.

## 3. Návrh funkcionality

Nejzákladnější funkcionalita programu je schopnost načíst data z XML souboru, tato data zpracovat do modelu a poté zobrazit formou váženého nesměrového grafu. Aby data byla zobrazena co nejpřehledněji, je využit layout algoritmus a jsou nastaveny velikosti elips (reprezentující uzly) a úseček (reprezentující hrany) podle významu.

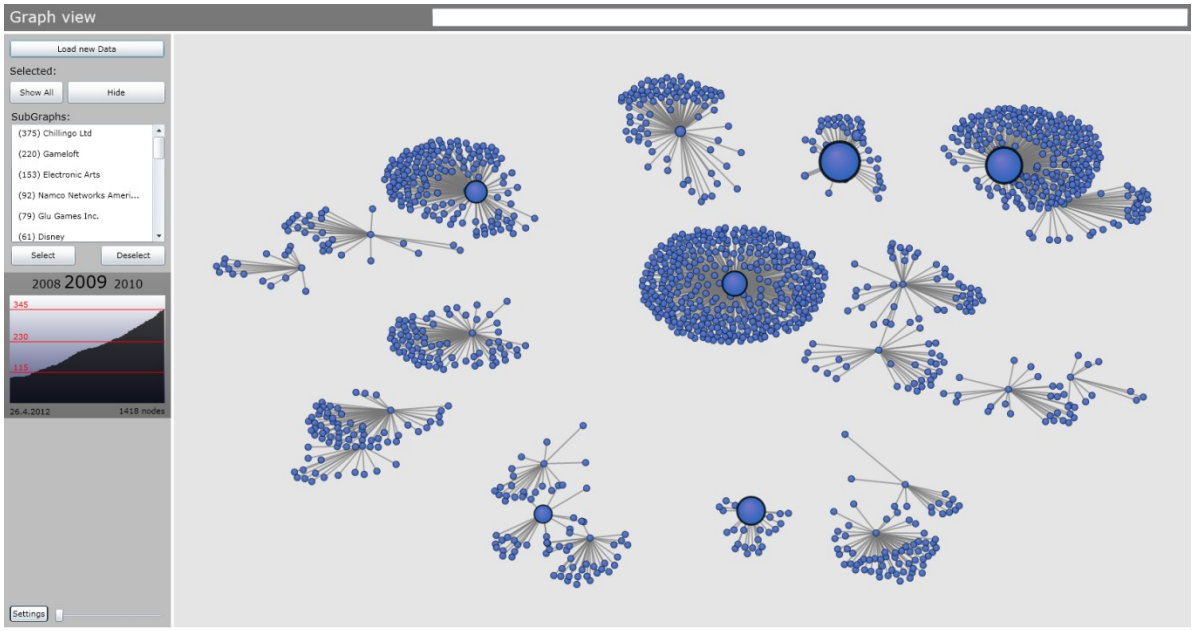
Dalšími významnými funkcemi programu je možnost libovolně označit uzly, s těmito uzly pohybovat po plátně, možnost skrýt uzly, odstranit uzly z databáze, nechat zobrazit příbuzné uzly vybraného uzlu a možnost nechat znovu automaticky rozložit graf na plátno.

Doplňující funkcí programu je možnost vybrat celé podgrafy, zobrazovat grafy v rámci vybraného data, možnost skrýt hrany s nízkými hodnotami, zvětšovat a zmenšovat uzly celého grafu a možnost vyhledávat uzly podle různých charakteristik.

## 4. Návrh GUI

### Úvod

Po spuštění programu se uživateli zobrazí uživatelské rozhraní. Toto rozhraní bylo vytvořeno jazykem XAML do podoby na obrázku níže. Vlevo nahoře se nachází tlačítko Load new Data. Toto tlačítko slouží k vybrání XML souboru, který obsahuje datovou strukturu. Po vybrání souboru a stisknutí tlačítka otevřít se automaticky načte a zobrazí graf. To má za následek naplnění 3 komponent.



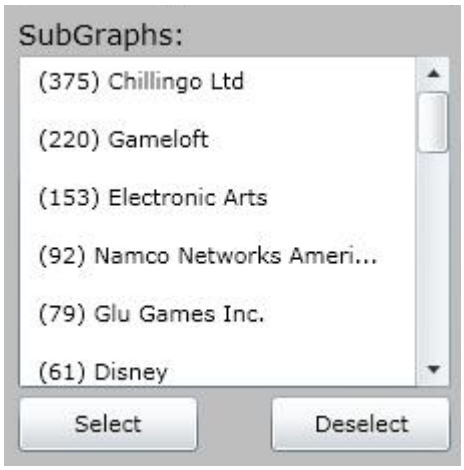

9 Obrázek uživatelského rozhraní aplikace.

## Plátno

První a nejviditelnější komponenta je plátno nacházející se v pravé části obrazovky. Na plátně se zobrazí modré kruhy reprezentující uzly a šedé přímky reprezentující hrany mezi uzly. Graf bude na plátně rozložen tak, aby byly uzly, které spolu nemají vztah co nejdál od sebe a zároveň uzly spojené hranou byly umístěny co nejblíže.

## Seznam podgrafů

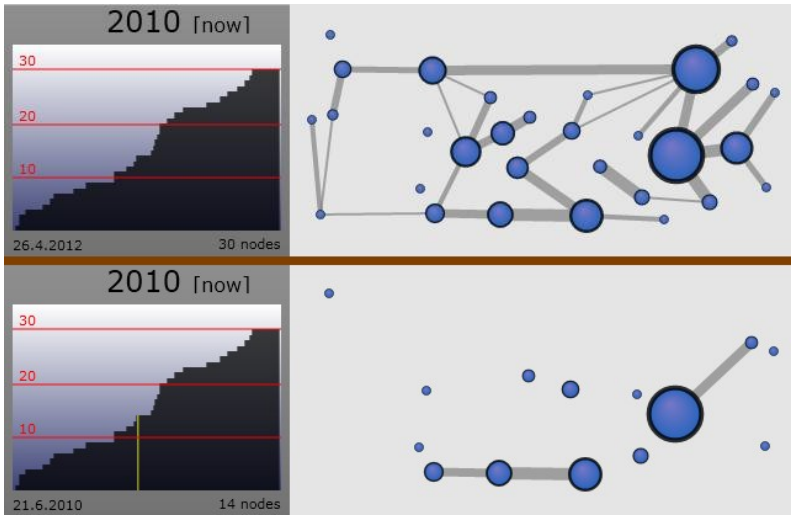
Druhá komponenta je seznam podgrafů. Po načtení dat se komponenta naplní jednotlivými podgrafy a počty uzlů, jež podgrafy obsahují. Tato komponenta slouží k hromadnému označení, či zrušení označení uzlů náležící vybranému podgrafu.




10 Detail na komponentu seznam podgrafů.

## Graf dle data

Třetí a poslední komponenta ovlivněná načtením dat je komponenta graf dle data. Tato komponenta zobrazuje počty uzlů k určitému datu. Tento počet je zobrazen vpravo dole. Současně vybrané datum je zobrazeno vlevo dole. Vrchní část komponenty slouží k vybrání roku, který nás zajímá. Levá hodnota je posun o rok do minulosti, střední hodnota je současně vybraný rok a pravá hodnota je posun o rok do budoucna. Relativní počet uzlů k danému datu můžeme také vidět výškou černé barvy na komponentě. Avšak nejdůležitější vlastností komponenty je, že při kliknutí a vybrání daného data se na plátně zobrazí pouze ty uzly, které v tomto datu již existovaly.




**11 Srovnání výřezu grafu při použití komponenty graf dle data.**

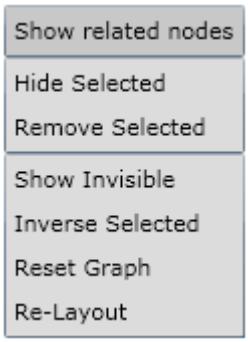
## Ostatní komponenty

Další komponenty již nejsou ovlivněny načtením grafu. Jedna taková komponenta se nachází vlevo dole na kontrolním panelu. Tuto komponentu představuje posuvník. Tento posuvník slouží k zneviditelnění hran s hodnotou nižší než je hodnota na posuvníku. Tato hodnota se nachází v rozmezí 1 až 100. Vpravo nahoře se zase nachází pole pro vyhledávání komponent. Tato komponenta může vyhledávat na základě spousty vlastností uzlů. Nevýhodou je, že vyhledávané výrazy musí mít pevně daný formát. Aby uživatel byl schopen s touto komponentou pracovat, existuje nápověda. Vlevo dole po kliknutí na tlačítko Settings se změní boční panel. Na tomto novém panelu se dole nachází kompletní nápověda pro vyhledávání. Na tomto panelu se dále nachází tlačítko pro návrat na původní panel a posuvník, který slouží k měnění velikosti uzlů a hran celého grafu.

## Kontextová nabídka

Jako poslední komponentu uživatelského rozhraní představím kontextovou nabídku. Kontextová nabídka se zobrazí kdykoli klikneme pravým tlačítkem na plátno. Obsah kontextové nabídky se ale mění v závislosti na okolnostech. Tlačítko Show related nodes se zobrazí pouze, pokud uživatel pravým tlačítkem klikl přímo na uzel. Toto tlačítko skryje celý graf a nechá zobrazené jenom uzly, spojené hranou s uzlem, na který uživatel klikl. Tlačítka Hide Selected a Remove Selected se zobrazí

pouze, pokud jsme vybrali alespoň jeden uzel. Hide selected schová vybrané uzly, zatímco Remove Selected uzly úplně odstraní z datové kolekce. Ostatní tlačítka jsou zobrazena vždy. Show Invisible zobrazí dříve skryté uzly, Inverse Selected označí dříve neoznačené uzly a naopak u dříve označených uzlů zruší výběr. Reset Graph vrátí do grafu všechny odstraněné uzly a jako poslední Re-layout umožňuje znovu nechat rozložit graf. Avšak rozloží se pouze uzly, které nebyly odstraněny z kolekce.




12 Kontextová nabídka se všemi tlačítky, jež lze zobrazit.

# 5. Konečná implementace

## Práce s uzly

Plátno i uzly jsou ze startu nastavené tak, aby reagovaly na událost stlačení levého tlačítka myši.

Pokud klikneme na plátno, funkce vyhodnotí podmínku, zda je stlačená klávesa shift. Pokud není klávesa shift stisknuta, funkce vynuluje kolekci označených uzlů. V obou případech dále uloží pozici, kde jsme klikli, nechá plátno kontrolovat událost pohybu kursoru a uvolnění myši a přestane reagovat na stlačené levé tlačítko. Dokud je stále stlačené tlačítko, plátno reaguje na pohyb kursoru, a při každém pohybu provede následující: Nejprve zajistí, aby výběrový čtverec nemohl opustit v jakémkoli směru plátno porovnáním pozice kursoru s velikostí plátna. Poté rozhodne na základě uložené počáteční hodnoty kursoru a aktuální hodnoty, kterým ze 4 směrů výběrové okno uživatel vytváří. Nyní vytvoříme na plátně čtverec podle dříve zjištěných souřadnic. Hned poté, co je čtverec vytvořen, zjistíme pro každý uzel v kolekci grafu, zda jsou souřadnice uzlu uvnitř výběrového čtverce. Pokud čtverec v minulosti byl v kolekci označených, ale již není a není stisknut shift, je uzel odstraněn z kolekce označených a okamžitě se provede překreslení pouze daného uzlu a jeho hran. Dále pokud uzel v kolekci nebyl a nyní je, přidá se uzel do kolekce označených uzlů a opět se provede překreslení pouze daného uzlu a jeho hran. Jakmile jsou označeny požadované uzly, zbývá uvolnit tlačítko myši. Při uvolnění tlačítka se z plátna zruší výběrový čtverec, plátno přestane kontrolovat pohyb kursoru a uvolnění levého tlačítka a opět začne hlídat stisk levého tlačítka.

Pokud klikneme na jakoukoli elipsu reprezentující uzel, funkce začne procházet souřadnice všech uzlů a hledá takový, který odpovídá souřadnicím kliknutí. Tyto souřadnice i první odpovídající uzel se uloží. Pokud jsme během stisku myši nedrželi shift, kolekce označených uzlů se smaže a nově se do ní vloží dříve nalezený uzel. Pokud byl shift stisknut, tak se pouze zajistí, aby kolekce uložených uzlů tento uzel obsahovala. V obou případech se pozměněné uzly překreslí. Dále se musíme připravit na možné posouvání uzlů. Proto pro kolekci označených uzlů zjistíme, jaké má hraniční hodnoty. Tyto

hodnoty slouží k tomu, aby při posouvání uzlů uzly neskončily mimo plátno. Tyto hodnoty uložíme, přestaneme sledovat stisk tlačítka a necháme elipsu kontrolovat pohyb kursoru a uvolnění stisku myši. Při každém pohybu kursoru zjistíme jeho souřadnice a ověříme, zda by se při tomto pohybu nedostaly označené uzly mimo plátno pomocí hraničních hodnot, jež jsme dříve získali. Nyní pro každý uzel z kolekce označených uzlů provedeme posun ke kurzoru. Pokud je však jedna z hraničních hodnot překročena, v tomhle směru k posunutí za hranici nedojde. Jakmile je dokončen posun, uvolněním levého tlačítka zrušíme u elipsy kontrolu pohybu kursoru a uvolnění myši a naopak opět zavedeme sledování stisku tlačítka.

## Načtení dat

Nejprve musí uživatel pomocí OpenFileDialog vybrat XML soubor s vhodnou datovou strukturou. Tento soubor je po načtení poslán funkci loadData v modelu. Tato funkce nejprve vymaže současné kolekce uzlů a hran pro případ, že by se jednalo již o druhé nahrání. Dále funkce z XML souboru roztřídí jednotlivé prvky na autory, aplikace a hrany a tyto prvky uloží do kolekcí. Každou z těchto kolekcí funkce dále zpracuje. Například u autorů vezme atributy id, jméno a datum přidání a tyto hodnoty předá funkci AddDeveloper. Tato funkce se postará, aby byl řádně vytvořen v modelu nový autor a byly provedeny všechny potřebné kroky pro některé ovládací prvky. Jakmile jsou vytvořené všechny uzly a hrany, provede se roztřídění do jednotlivých subgrafů a chronologické setřídění.

Jakmile je tohle provedeno, jsou ve view promazány kolekce reprezentující grafické zobrazení grafu. Z modelu jsou načteny uzly podle typu a poté jsou dále zpracovány. Pro každý uzel je vytvořena jeho grafická reprezentace a poté je předáno jeho id. Zobrazovaným uzlům je nastavený tooltip podle hodnot v modelu a poté je uložen do kolekce v grafu. Totéž je provedeno s hranami. Nyní je třeba rozestavit uzly do prostoru, proto se na celý graf použije layout funkce. Jakmile je layout dokončen, je třeba upravit velikosti uzlů a hran podle důležitosti. Toto se provede ve funkci setSizes. Nyní zbývá již pouze vykreslit graf.

## Nastavení velikostí uzlů

Funkce setSizes nejprve vezme kolekci hran v grafu, a zjistí jaká je maximální hodnota hrany. Poté každou hranu vynásobíme deseti a vydělíme dříve získanou maximální hodnotou. Avšak pokud je výsledek menší než 2, použijeme hodnotu 2. Tuto hodnotu nastavíme jako tloušťku čar reprezentující hrany. Nyní je třeba upravit velikosti uzlů. Nejprve musíme zjistit maximální hodnotu mezi uzly. Hodnotu uzlu vypočítáme sečtením hodnot všech hran, jež danému uzlu náleží. Nyní již víme maximální hodnotu, takže pro všechny uzly určíme průměr elips vynásobením získané hodnoty 60ti a posléze vydělením maximální hodnotou. Výsledný průměr elipsy však nemůže být nižší než 10.

## Rozdělení do podgrafů

V modelu kromě dat samotných taky existuje třída Subgraphs. Tato třída obsahuje metody, které jsou volány pokaždé, je-li do modelu přidán nový uzel nebo hrana. V případě, že se jednalo o uzel, vytvoří se nová instance třídy Subgraph, která obsahuje jméno podgrafu, id uzlů daného podgrafu a několik základních metod. Do této nové prázdné instance je přidáno id uzlu a poté je instance vložena do kolekce podgrafů.

Pokud však byla přidána hrana, metoda nejprve zjistí, zda uzly, které hrana spojuje, již náleží do některého podgrafu. Pokud ne, vytvoří nový podgraf pro uzel, jenž v žádném podgrafu nebyl. Poté je provedeno propojení podgrafů pomocí metody MergeSubgraphs. To proběhne tak, že z jednoho podgrafu postupně převezmeme veškeré id, které obsahoval, a poté je prázdný podgraf odstraněn.

Nakonec ještě může být kolekce podgrafů setříděna podle počtu id, jenž podgrafy obsahují.

## Vykreslení grafu

Jakmile máme k dispozici všechna data i jejich reprezentaci ve view, zbývá tato data vykreslit. K tomuto slouží třida GraphDrawer. Třídě předáme jako parametry graf a plátno, na které chceme tento graf vykreslit. Nejprve je u všech uzlů a hran rozhodnuto, jestli jsou viditelné, vybrané nebo jinak ovlivněny. Podle toho se u uzlů vybere, který štětec se použije k jejich vykreslení. Nyní se elipsám nastaví souřadnice na plátně, které uzly uchovávají jako proměnné. Poté jsou tyto uzly v případě prvního vykreslení vloženy na plátno. Nakonec jsou vykreslovány hrany. Hrany jsou vykresleny v podstatě stejně jak uzly, akorát se nastavují souřadnice úseček místo elips.

## Kontextová nabídka

Kontextová nabídka se zobrazuje, pokud uživatel klikne pravým tlačítkem na plátno. Po kliknutí se uloží souřadnice kurzoru. Pokud již byla otevřená kontextová nabídka, tak se zruší. Nyní se vytvoří nová nabídka. Abychom zabránili dalším manipulacím s uzly, dokud nabídku neuzavřeme, přes plátno necháme vytvořit druhé plátno, které bude přebírat události vytvořené mačkáním tlačítek myši.

Kontextová nabídka se skládá ze 3 nabídek a v nich7 tlačítek. Tato tlačítka ovládají metody zobraz příbuzné uzly, skryj vybrané uzly, odstraň vybrané uzly, prohoď vybrané a nevybrané uzly, zobraz neviditelné uzly, vrať všechny uzly do grafu a znovu rozložit graf.

Nejprve však musíme vytvořit schránku, do které tohle všechno vložíme. Od toho si vytvoříme stackpanel. Nyní je třeba do stackpanelu vložit nabídky a do nich tlačítka. Avšak ne každé tlačítko bude nakonec v kontextové nabídce umístěno. Metoda vyhodnotí několik podmínek a podle toho rozhodne, která tlačítka zobrazí.

První podmínkou je podmínka, kde se ptáme, zda jsou nějaké uzly zneviditelněné. Pokud žádné uzly neviditelné nejsou, nemá smysl zobrazit tlačítko odhalit neviditelné. Druhá podmínka se ptá, zda uživatel klikl přímo na uzel. V případě že neklikl, není důvod nabízet příbuzné uzly. Poslední podmínka je dotaz, zda jsou vybrány nějaké uzly. Pokud tato podmínka není splněna, nezobrazí se metody skryj vybrané uzly a odstraň vybrané uzly.

Nyní je kontextová nabídka vytvořena a zobrazena, čeká se pouze na reakci uživatele. Kontextová nabídka se uzavře, klikneme-li na jedno z tlačítek v nabídce nebo klikneme levým tlačítkem na plátno.

# 6. Výkonnostní testy

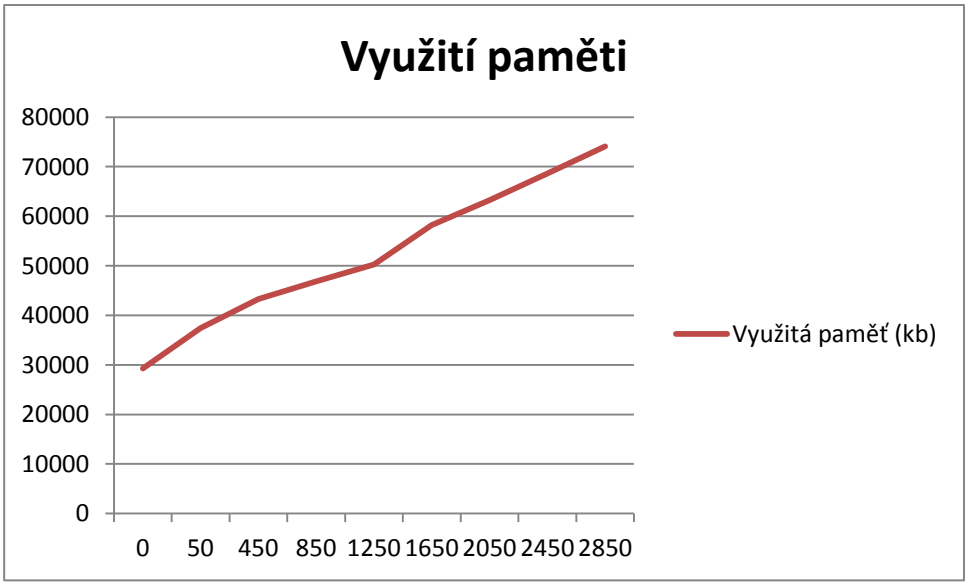
Pro vyhodnocení použitelnosti programu bylo provedeno několik měření rychlosti provedení úkonů. Tyto testy byly provedeny na starší počítačové sestavě Micro-star international MS-7125, procesor AMD Athlon(tm) 64, 2010Mhz, 1gb RAM. Mezi provedené testy patří paměťová náročnost, rychlost načtení dat představující uzly, rychlost načtení dat představující hrany, rychlost přesunu většího množství uzlů, rychlost přesunu jednoho uzlu s velkým množstvím hran a testy rychlosti layout pro uzly i hrany.

## Využití paměti

Měření bylo provedeno v klidu po načtení různě velkých kolekcí. Kolekce měly vždy 50 hran

| Číslo měření | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Počet uzlů | 0 | 50 | 450 | 850 | 1250 | 1650 | 2050 | 2450 | 2850 |
| Využitá paměť (kb) | 29258 | 37426 | 43278 | 46810 | 50290 | 58178 | 63222 | 68590 | 74074 |

1 Tabulka využití paměti v závislosti na počtu uzlů.



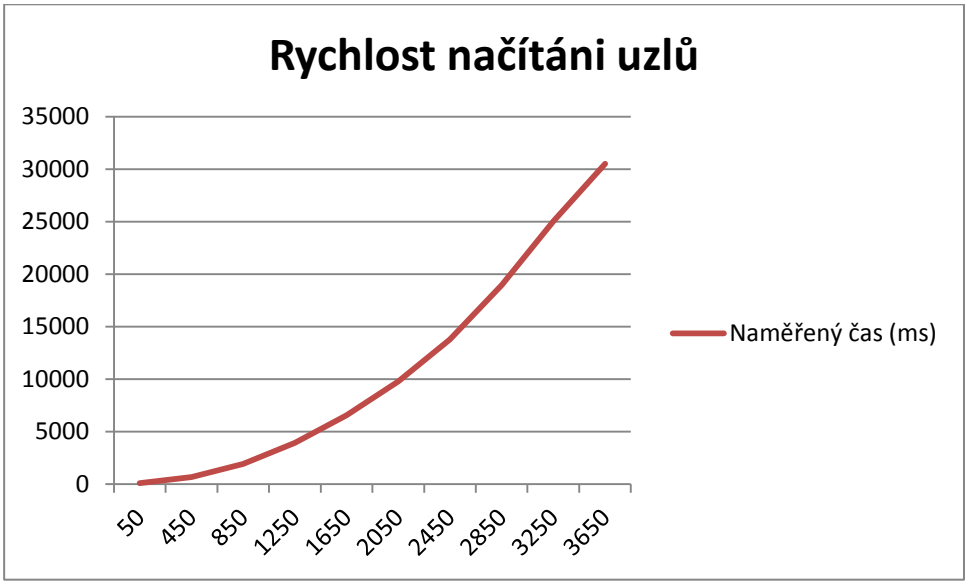

13 Graf využití paměti v závislosti na počtu uzlů.

## Načtení uzlů

Jedná se o dobu od vybrání datové kolekce pomocí výběrového okna po konečné zobrazení dat v komponentách a grafu na plátně. Načítání dat trvá relativně dlouhé časové úseky, avšak po načtení dat již málokdy potřebujeme opět načíst data, tudíž delší načítání lze ignorovat. V tabulce se nachází časy pro testovací datové kolekce. V těchto kolekcí je vždy pouze 50 hran a mění se pouze počet uzlů.

| Číslo měření | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Počet uzlů | 50 | 450 | 850 | 1250 | 1650 | 2050 | 2450 | 2850 | 3250 | 3650 |
| Naměřený čas (ms) | 93,75 | 671,8 | 1921 | 3953 | 6578 | 9796 | 13796 | 18953 | 25015 | 30500 |

2 Tabulka rychlosti načtení uzlů.



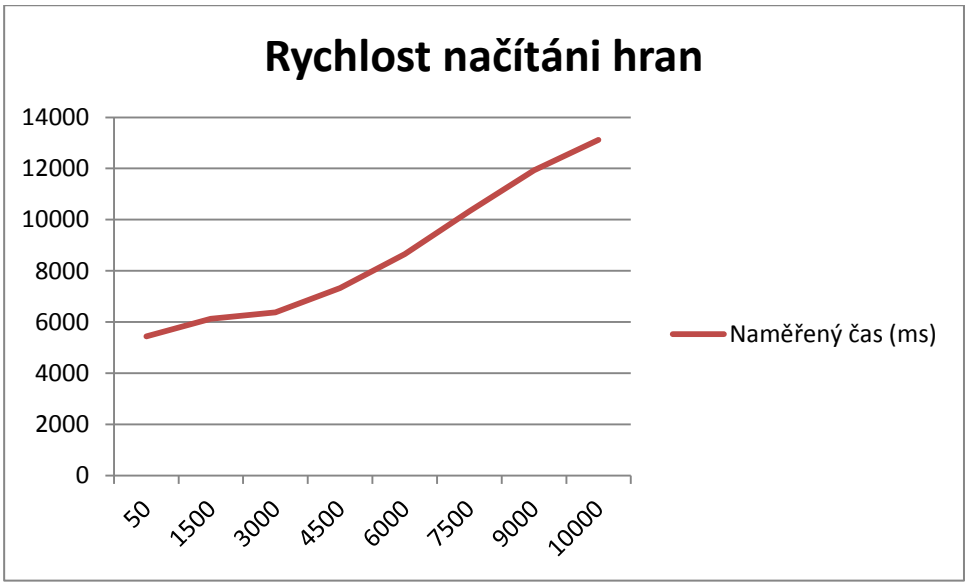

14 Graf rychlosti načtení uzlů.

## Načtení hran

Jedná se opět o dobu vybrání datové kolekce pomocí výběrového okna, po konečné zobrazení dat v komponentách a grafu na plátně. Podle výsledku měření je doba načítání datové kolekce daleko méně ovlivněna počtem hran, než počtem uzlů. Pro měření byly použity kolekce s 1500 uzly a různými počty hran.

| Číslo měření | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Počet hran | 50 | 1500 | 3000 | 4500 | 6000 | 7500 | 9000 | 10000 |
| Naměřený čas (ms) | 5437 | 6125 | 6375 | 7328 | 8656 | 10328 | 11937 | 13125 |

3 Tabulka rychlosti načtení hran.



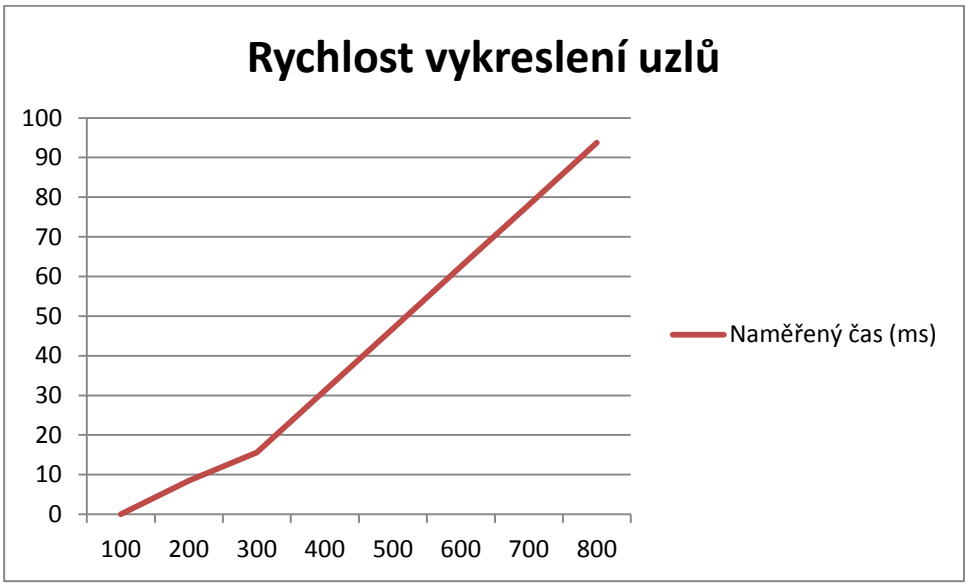

15 Graf rychlosti načtení hran.

## Rychlost přesunu uzlů

Jednou z důležitých funkcí komponenty je možnost libovolně přesouvat označenou kolekci uzlů. Avšak existuje limit, kdy již posun uzlů není vykreslován správně, nebo nefunguje téměř vůbec. Hlavní problém Přesouvání uzlů spočívá v tom, že během přesunu uzlů se všechny vybrané uzly musí neustále vykreslovat, dokud uživatel nezvolí vhodnou pozici, kde chce uzly umístit. Cílem tohoto měření je zjistit dobu vykreslení kolekce označených uzlů. Tato operace je však prováděna při přesunu neustále, proto bude následovat slovní ohodnocení jednotlivých měření. Testovací kolekce neobsahovala žádné hrany.

| Číslo měření | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Počet uzlů | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 |
| Naměřený čas (ms) | 0 | 8,5 | 15,6 | 31,25 | 46,85 | 62,5 | 78,13 | 93,75 |

4 Tabulka rychlosti vykreslení uzlů.




16 Graf rychlosti vykreslení uzlů.

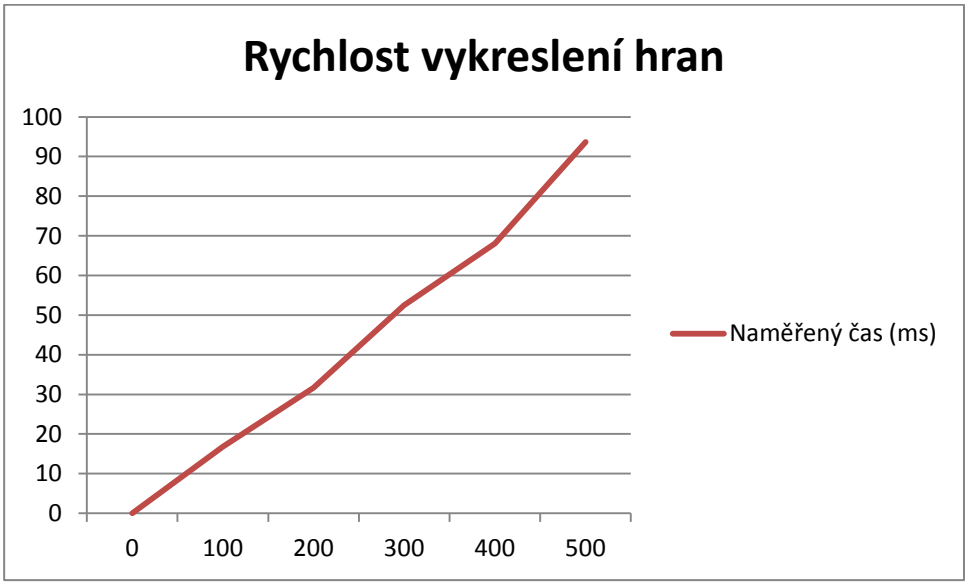
Z grafu je patrné, že se jedná o lineární závislost. Během prvních dvou měření bylo přesouvání uzlů naprosto bez prodlevy i při velmi rychlém pohybu kurzoru. Při dalším měření se začaly projevovat prodlevy, pokud se s kurzorem pohybovalo příliš rychle, ale stále v rámci subjektivní tolerance. Situace se během dalších měření dále zhoršovala a pro čisté zobrazení náhledu jsem byl nucen s kurzorem pohybovat minimální rychlostí. Během 7. měření se situace výrazně zhoršila. Vykreslovat 700 uzlů několikrát za vteřinu moje sestava už vůbec nezvládala, a pokud jsem chtěl vidět uzly, s kterými jsem pohyboval, musel jsem s kurzorem zastavit na místě. Při posledním měření již docházelo k prodlevám, i když byl po pohybu kurzor zastaven. Pracovat v tomto stavu bylo téměř nemožné. Pro práci s grafy bych na mé sestavě doporučil limit maximálně přemisťovat 300 uzlů pro čistotu pohybu. Přemístit lze najednou i větší množství uzlů, avšak náhled již má velké zpoždění, což ovlivňuje pracovní výkon uživatele.

## Rychlost přesunu hran

Nyní se zaměříme na rychlost vykreslování 1 uzlu, kterému náleží velké množství hran. Opět jako v předešlém případě, i při tomto měření zjišťujeme čas potřebný k vykreslení vybraného uzlu, které probíhá neustále několikrát za sekundu, během přesunu uzlu po plátně. Testovací kolekce obsahovala 501 uzlů. Slovní hodnocení opět pod tabulkou.

| Číslo měření | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Počet hran | 0 | 100 | 200 | 300 | 400 | 500 |
| Naměřený čas (ms) | 0 | 16,8 | 31,7 | 52,5 | 68,1 | 93,7 |

**5 Tabulka rychlosti vykreslení hran.**




**17 Graf rychlosti vykreslení hran.**

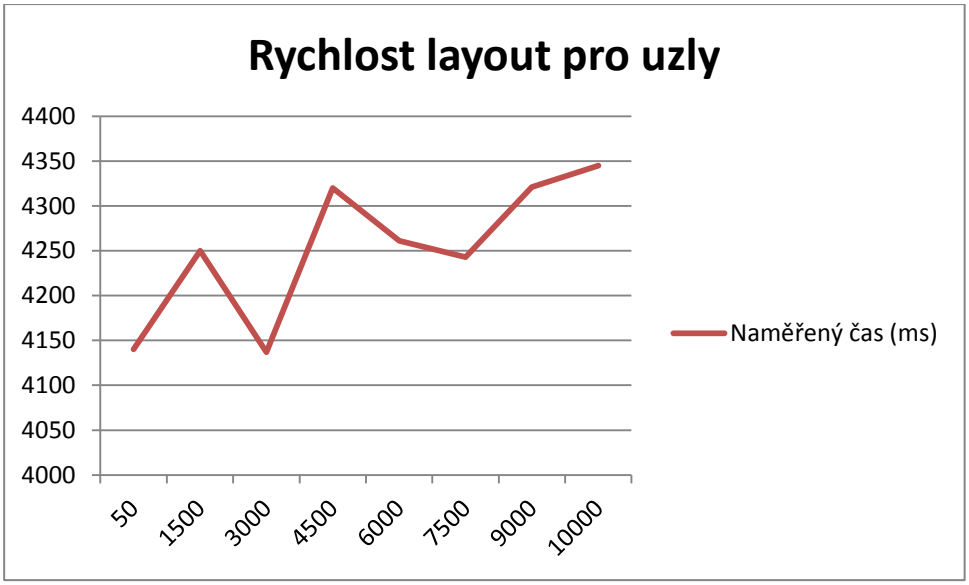
Podle naměřených údajů je patrné, že je výrazně pomalejší vykreslovat hrany než uzly. První měření bylo pro porovnávací účely pohyb uzlem beze hran. Tento pohyb byl samozřejmě naprosto plynulý a vykreslení proběhlo okamžitě. Druhé měření bylo stále ještě plynulé a bez problémů. Během třetího měření se již začaly objevovat prodlevy avšak stále v toleranci. Po čtvrtém měření již musím výrazně zpomalit pohyb kurzoru pro čisté vykreslování náhledu. Během pátého měření již pro vykreslování náhledu musím počkat s kurzorem na místě. Poslední měření ukázalo, že na této sestavě je pohyb s uzlem vlastnícím 500 hran neovladatelný, vykreslování uzlu probíhá až po 2 sekundách po zastavení kurzoru. Pro ještě rozumný přesun uzlu s mírným opožděním náhledu bych doporučil posouvat uzly do 300 hran. To však neznamená, že nemůže být přesunut uzel obsahující 1000 hran, pouze náhled nebude fungovat.

## Rychlost layout pro kolekci uzlů

Cílem měření je zjistit, jak dlouho trvá layout pro různě velké kolekce uzlů. Pro zkušební kolekce je pevně stanoven počet hran na 50, liší se tedy pouze počty uzlů. Podle naměřených hodnot je jasné, že počet uzlů silně ovlivňuje rychlost layout. Vzhledem k tomu, že se layout používá jen při načítání dat a během re-layout grafu, jsou tyto hodnoty dostatečně nízké vzhledem k množství uzlů, které layout zpracovává

| Číslo měření | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Počet uzlů | 50 | 450 | 850 | 1250 | 1650 | 2050 | 2450 | 2850 |
| Naměřený čas (ms) | 0 | 375 | 1312 | 3250 | 5468 | 7593 | 11906 | 15750 |

**6 Tabulka rychlosti layout pro uzly.**



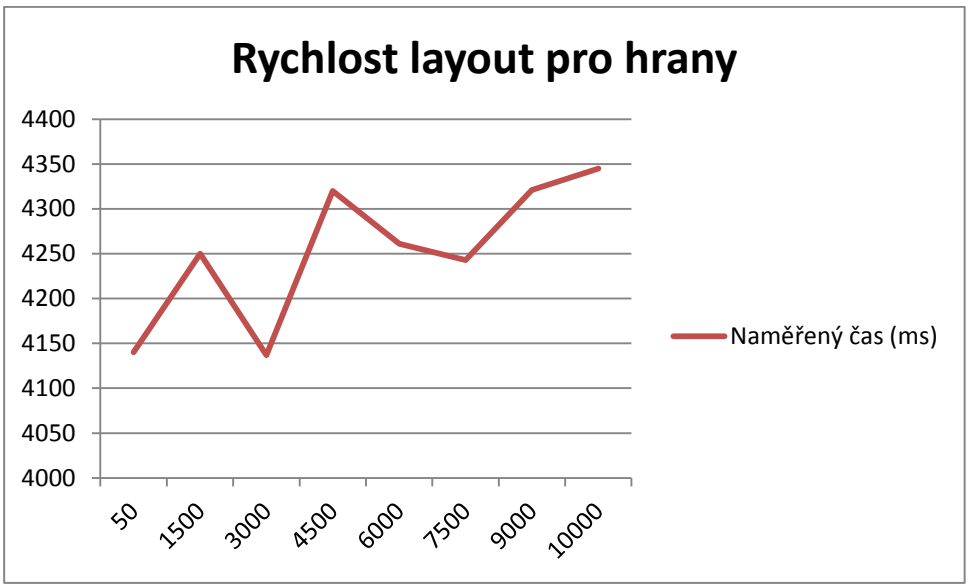

**18 Graf rychlosti layout pro uzly.**

## Rychlost layout pro kolekci hran

Podobně jako v minulém případě, toto měření má za úkol zjistit, jak moc ovlivňuje rychlost layout množství hran v grafu. Testovací kolekce obsahují vždy 1500 uzlů a různé počty hran. Z výsledku je patrné, že pro layout algoritmus nehraje množství hran žádnou roli. Odchylky měření byly pravděpodobně způsobeny programy na pozadí.

| Číslo měření | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Počet hran | 50 | 1500 | 3000 | 4500 | 6000 | 7500 | 9000 | 10000 |
| Naměřený čas (ms) | 4140 | 4250 | 4137 | 4320 | 4261 | 4243 | 4321 | 4345 |

7 Tabulka rychlosti layout pro hrany.




19 Graf rychlosti layout pro hrany.

# 5. Závěr

Záměrem práce bylo vytvořit program vhodný pro vizualizace datových kolekcí. Vytvořený program dokáže zpracovat a docela přehledně zobrazit kolekce o tisících uzlů a hran. Program dále umožňuje mezi zobrazenými daty filtrovat, vyhledávat, různě posouvat uzly a sledovat postup vzniku datové kolekce. Program zvládá na obstojné úrovni práci s velkým množství uzlů a hran. Toto množství se samozřejmě liší podle počítačové sestavy uživatele, v mém případě se jednalo o posun 300 uzlů, nebo posun uzlu náležícímu 300 hran.

Během práce jsem se seznámil s vizualizací grafů, layout algoritmy a především jsem se naučil programovat v prostředí Silverlight. Zároveň jsem se zdokonalil v programování v jazyku c# a .net Framework.

## Možnosti dalšího rozšíření

Jako možnost dalšího rozšíření bych viděl zavedení více druhů layout algoritmů, které by umožnovaly pro různé datové kolekce lepší výsledné zobrazení nebo rychlejší zpracování dat. Dále by mohla být zavedena podpora směrových grafů a grafické odlišení jednotlivých druhů uzlů. Nakonec by ještě mohla být přidána možnost editace dat pomocí uživatelského rozhraní.

# Literatura

[1] NAVARRO, Gonzalo. A guided tour to approximate string matching: algorithms for the visualization of graphs. Vyd. 1. Překlad Karel Voráček. Upper Saddle River, N.J.: Prentice Hall, c1999, 397 s. ISBN 10.1145/375360.375365

[2] DI BATTISTA, Giuseppe. Graph drawing: algorithms for the visualization of graphs. Vyd. 1. Překlad Karel Voráček. Upper Saddle River, N.J.: Prentice Hall, c1999, 397 s. ISBN 01-330-1615-3.

[3] LACKO, Ľuboslav. *Silverlight*: *výukový průvodce tvorbou interaktivních aplikací*. Vyd. 1. Brno: Computer Press, 2010, 464 s. ISBN 978-80-251-2716-2.

[4] PETZOLD, Charles. Programování Microsoft Windows Forms v jazyce C#. Vyd. 1. Překlad Karel Voráček. Brno: Computer Press, 2006, 356 s. ISBN 80-251-1058-3.

[5] CHRIS, Solomon a Toby BRECKON. Fundamentals of Digital Image Processing: a practical approach with examples in matlab. 1. vyd. Oxford: Wiley-Blackwell, 2011, 328 s. ISBN 978-0-470-84473-1.

[6] LIOTTA, Giuseppe. Graph drawing: 11th international symposium, GD 2003, Perugia, Italy, September 21-24, 2003 : revised papers. New York: Springer, 2004, 539 s. ISBN 35-402-0831-3.

[7] EDITOR, Gillian Dobbie. 11th international symposium, GD 2003, Perugia, Italy, September 21-24, 2003 : revised papers. Darlinghurst, Australia: Australian Computer Society, Inc, 2007, 539 s. ISBN 1-920-68243-0.

[8] EDITOR, Gillian Dobbie. Graph drawing: 18th international symposium, GD 2010, Konstanz, Germany, September 21-24, 2010. revised selected papers. 1st ed. New York: Springer, 2011, 539 s. ISBN 36-421-8468-5.

[9] NORTH, Stephen C. Graph drawing: Symposium on Graph Drawing, GD '96, Berkeley, California, USA, September 18-20, 1996 : proceedings. 1st ed. New York: Springer, c1997, 408 s. ISBN 35-406-2495-3.

[10] GRUNDSPENKIS, Janis. Advances in databases and information systems: associated workshops and doctoral consortium of the 13th East European Conference, ADBIS 2009, Riga, Lativia, September 7-10, 2009 : revised selected papers. 1st ed. New York: Springer, c2010, 256 s. Lecture notes in computer science, 5968. ISBN 36-421-2081-4.

[11] GRUNDSPENKIS, Janis. Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000: September 27-29, 2000, A Curuña, Spain : proceedings. 1st ed. Los Alamitos, Calif.: IEEE Computer Society, c2000, 261 s. Lecture notes in computer science, 5968. ISBN 978-076-9507-484.

[12] WEGNER, Peter. A technique for counting ones in a binary computer: September 27-29, 2000, A Curuña, Spain : proceedings. 1st ed. Los Alamitos, Calif.: IEEE Computer Society, c2000, 261 s. Lecture notes in computer science, 5968. ISBN 10.1145/367236.367286. Dostupné z: http://portal.acm.org/citation.cfm?doid=367236.367286

[13] GUSFIELD, Dan. Algorithms on strings, trees, and sequences: computer science and computational biology. 1st ed. New York: Cambridge University Press, 1997, 534 s. Lecture notes in computer science, 5968. ISBN 05-215-8519-8. Dostupné z: http://portal.acm.org/citation.cfm?doid=367236.367286

# Adresářová struktura přiloženého DVD

| | |
|---|---|
| /Program | Zde se nachází kód ukázkové aplikace |
| /Text | Složka s pdf verzí tohoto dokumentu |
| /Ukázka | Složka s odkazem na internetové stránky, na kterých se nachází aplikace |
| /Kolekce | Zde naleznete ukázkovou datovou kolekci pro aplikaci |